МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья поиска и алгоритмы сжатия
Вариант 1

Студент гр. 8304 _____ Щука А. А.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2019

Цель работы.

Научиться применять бинарные деревья для решения задач кодирования (сжатия) и поиска

Постановка задачи.

Вариант 1: с помощью бинарного дерева реализовать кодирование: Фано-Шеннона

Описание алгоритма.

Пусть набор весов W упорядочен, а именно: $w_1 \geq w_2 \geq \ldots \geq w_n$. В качестве корня дерева выбирается такой узел (и соответственно набор $W_{1\ldots n}$ разбивается на два поднабора $W_{1\ldots k}$ и $W_{k+1\ldots n}$ так), что веса поддеревьев различаются минимально. Алгоритм поиска такого узла: перебираем узлы до тех пор, пока разница между весами левой части набора и правой не будет больше либо равна весу следующего узла. С левой стороны от корня должно стоять поддерево с меньшим весом, а с правой – с большим. Эта процедура повторяется для поддеревьев до тех пор, пока не будет получен лист в качестве текущего поддерева.

Для кодирования сообщения необходимо идти по дереву от листа со значением нужного символа к корню дерева. Если текущий узел находится от родителя с левой стороны, то выводим 0, если с правой – 1. Таким образом мы можем получить код сообщения по бинарному дереву, закодированному с помощью алгоритма Фано-Шеннона.

Спецификация программы.

Программа предназначена для кодирования сообщения методом Фано-Шеннона.

Программа написана на языке C++. Входными данными являются символы, которые необходимо закодировать, считываемые из файла или вводимые с клавиатуры. Выходными данными являются промежуточные значения и конечный результат. Данные выводятся на экран монитора и записываются в файл result.txt. Результат работы программы в среде Windows в консоли с кодировкой

Windows-1251 представлен на рис. 1. Исходный код программы приведен в приложении В.

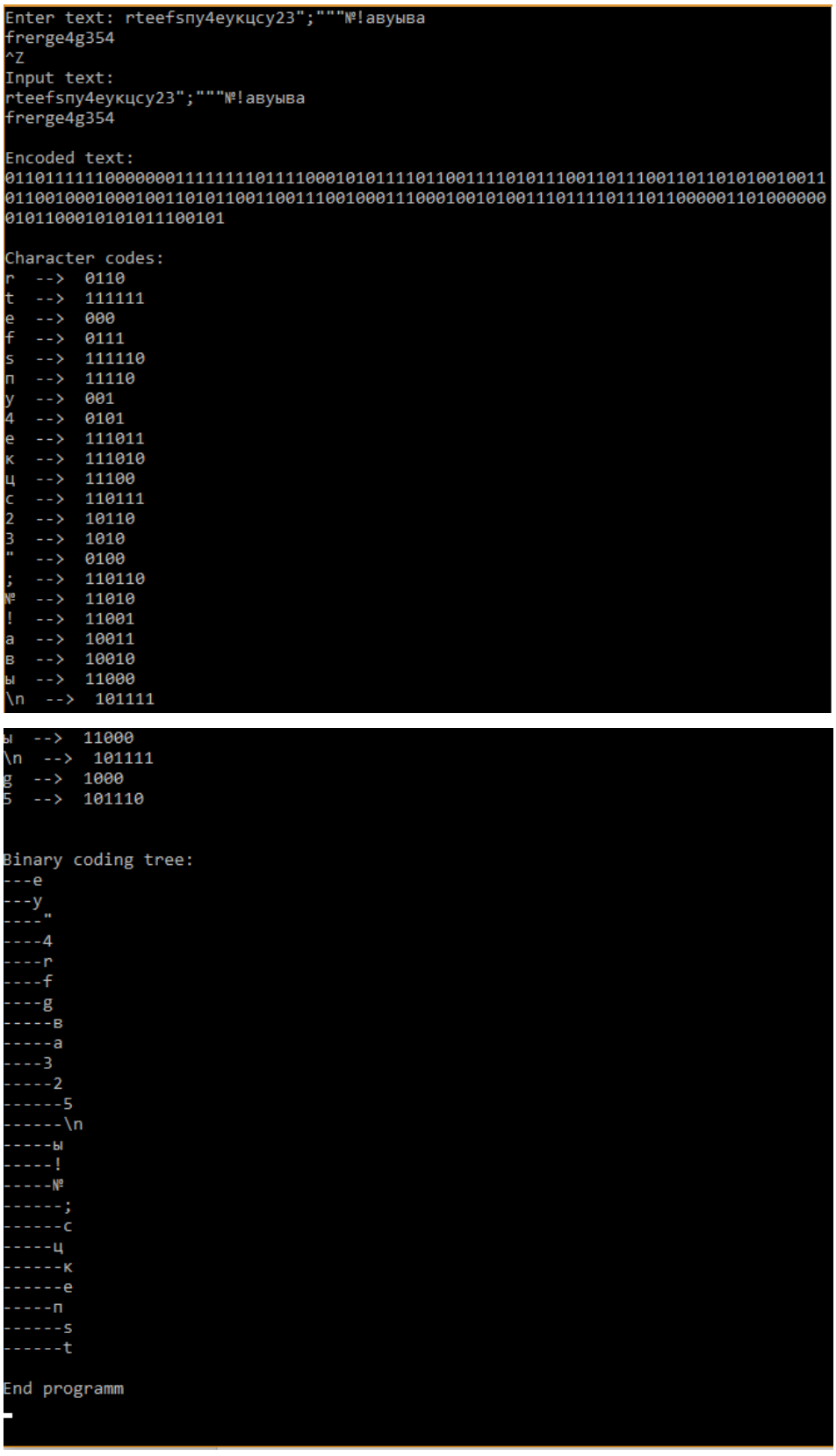

Рисунок 1 – Результат работы программы

Тестирование программы.

Тестирование программы приведено в приложении А (содержимое файла result.txt).

Описание СД и методов для работы с ними.

`struct Symbol`

Структура для хранения символа и его 'веса'.

Поля структуры:

```
char data_;
size_t weight_;
```

`class EncodeTree`

Базовый абстрактный класс дерева, реализующий методы для работы с бинарным деревом кодирования. Экземпляр класса является узлом дерева.

Поля класса:

```
Symbol data_;
EncodeTree* parent_;
EncodeTree* left_;
EncodeTree* right_;
```

Getter's – методы:

```
const EncodeTree* getLeft() const;
const EncodeTree* getRight() const;
const EncodeTree* getParent() const;
const Symbol& getData() const;
```

Setter's – методы:

```
void setLeft(EncodeTree* left);
void setRight(EncodeTree* right);
void setParent(EncodeTree* parent);
void setData(Symbol data);
```

Метод объединения левого и правого поддеревьев в узел:

```
void concat(size_t weight, EncodeTree* left, EncodeTree* right);
```

Метод для кодирования строки с помощью бинарного дерева кодирования:

```
std::string encode(const std::string& message);
```

Два виртуальных метода для создания дерева. В зависимости от алгоритма, класс-наследник реализует эти методы:

```
virtual void createEncodeTree(const std::string& message) = 0;
```

```
virtual void createEncodeTree(std::map<char, size_t>& symbolMap,
                              std::vector<char>& symbolVector,
                              size_t left, size_t right, size_t sum) = 0;
```

Метод поиска листа дерева с заданными данными:

```
const EncodeTree* findSymbol(char data) const;
```

Метод записи дерева в строку в виде уступчатого списка

```
void printTree(std::string& result, size_t level = 0) const;
```

Метод создания строки кодов символов:

```
std::string createCodeSymbols(const std::string& message) const;
```

class FanoShannonTree

Класс-наследник EncodeTree, реализующий алгоритм кодирования Фано-Шеннона с помощью бинарного дерева кодирования.

Два метода создания дерева кодирования с помощью алгоритма Фано-Шеннона

```
void createEncodeTree(const std::string& message);

void createEncodeTree(std::map<char, size_t>& symbolMap,
                      std::vector<char>& symbolVector,
                      size_t left, size_t right, size_t sum);
```

Статический метод для получения середины массива символов, отсортированных по не возрастанию, по их 'весу'

```
static size_t getMiddle(std::map<char, size_t>& symbolMap,
                        std::vector<char>& symbolVector,
                        size_t left, size_t sum, size_t& leftSum,
                        size_t& rightSum);
```

Выводы.

В ходе работы были изучены способы создания бинарных деревья с помощью алгоритма Фано-Шеннона и кодирования сообщений.

Приложение А. Содержимое файла result.txt

```
Read text from file ./Tests/test1.txt
Input text:
gsfughuhgkjhKJGHIUKHKNHIguynjgkhnYGbuybUNiuk
RUKYnuikhmhiduhniuehn ugNIygniygmmmiuu gGNIy
URYiuni UIOUyniu yuIUt yrtiol

Encoded text:
010011111111111111100000100010100001010100110001110000101101101111110110111110110100110
001011011011010110101111101101001010000001110110111000010011000010101101100111011111100
100001111110011000101110010001100011010111010100010110110010110000000111000010110101010 1
001111110100001010110001000111110001010110101000000100101111001011101000110001011101001
010110101101010010000001010001001101111011110010111110101000111010110010010000110001101
001000100111111010100001110110001000101000111000100110001110111010001111111011111011001 1
111001111100011010

Character codes:
g  -->  0100
s  -->  11111111
f  -->  11111110
u  -->  000
h  -->  0101
k  -->  11000
j  -->  111000
K  -->  10110
J  -->  1111110
G  -->  110111
H  -->  110110
I  -->  1001
U  -->  1000
N  -->  10111
y  -->  0111
n  -->  0110
Y  -->  11001
b  -->  111001
i  -->  001
\n  -->  11010
R  -->  111010
m  -->  10101
d  -->  1111101
e  -->  1111100
space  -->  10100
O  -->  1111010
t  -->  111011
r  -->  1111011
o  -->  1111001
l  -->  1111000

Binary coding tree:
---u
---i
----g
```

```
----h
----n
----y
----U
----I
-----space
-----m
-----K
-----N
-----k
-----Y
-----\n
------H
------G
------j
------b
------R
------t
-------l
-------o
-------O
-------r
-------e
-------d
-------J
--------f
--------s

Read text from file ./Tests/test2.txt
Input text:
(*&^%^88%5^%43@@3$%67* &^45*90(*7^%4#23$%67*90_)(8&6543
      &^7*9)(&62#$5^78(0;{"/;{;>/"{:/"[]"/;['?"

Encoded text:
01010001001100000010000010101101010010101000000001010011100101110101110101001010110001
10000110000111110100011000010011101000001101111100001010001011000000010100111101101110
01001010110001010000110000110111111000111100110111010110101001110001010010011100101110110
11101110011000001100001101111101110101001110001110011011010110101000000011010101010111
00011101100101000111101110110010111011110110111101001100111111000111101001101011111010
00011110111011010111111011111110100

Character codes:
(  -->  0101
*  -->  0001
&  -->  0011
^  -->  0000
%  -->  0010
8  -->  10101
5  -->  10100
4  -->  10011
3  -->  10010
@  -->  111010
$  -->  10110
```

```
6  -->  1000
7  -->  0110
space  -->  1111010
9  -->  10111
0  -->  11000
#  -->  110110
2  -->  11100
_  -->  111100
)  -->  110111
\n  -->  1110110
tabulation  -->  1110111
;  -->  01110
{  -->  11001
"  -->  0100
/  -->  01111
>  -->  1111011
:  -->  1111100
[  -->  11010
]  -->  1111101
'  -->  1111110
?  -->  1111111

Binary coding tree:
----^
----*
----%
----&
----"
----(
----7
-----;
-----/
----6
-----3
-----4
-----5
-----8
-----$
-----9
-----0
-----{
-----[
------#
------)
-----2
------@
-------\n
-------tabulation
------_
-------space
------->
-------:
-------]
```

```
-------'
-------?

Read text from file ./Tests/test3.txt
Input text:
gdyutrfb876bewtbf76iyT&6tngi8Ubgiyni   b utbGi6uytibU
&o*NPmo8&nb6&Nt9b 87b6i89b7o^up8(!N8u1Upm8Pbopb1up9Nmo
&b&o6b(*^7NyojUno*o
IYB*(&^nm7(**Pop7(N8b6
&^B9

Encoded text:
11011011111111101010111110000111111101110110000001101011010000011111101111101100000001
10110010110100011010101111110001010010010000101001101100110001110110000110110011010101
01000110111011100011101001111100000001111011101100100011111010110000011000010110110000
01000101000101110110111110100010001101010101000000100010100111010000110010001110100011
10110000100011000111100100001011001010111011111001100111001011110010111000110111111000
10110100111101000111101110000010100110001110000111110011110010111011010001011000010100
00101000100100000100101000110111010110111010101001011110110101101010000101000100101100
11110101111000111001100011001001010101111010011010010111001010001100011101110010100110
01110010011100011000010011000010101011111100111001
Character codes:
g  -->  110110
d  -->  11111111
y  -->  10101
u  -->  01111
t  -->  10000
r  -->  11111110
f  -->  1110110
b  -->  000
8  -->  0011
7  -->  01011
6  -->  0100
e  -->  1111110
w  -->  1111101
i  -->  0110
T  -->  1111100
&  -->  01010
n  -->  10100
U  -->  10110
tabulation  -->  1110111
space  -->  111010
G  -->  11110111
\n  -->  11000
o  -->  0010
*  -->  10001
N  -->  01110
P  -->  110111
m  -->  11010
9  -->  11001
^  -->  10111
p  -->  10011
(  -->  10010
```

```
!  -->  1111001
1  -->  111000
j  -->  11110110
I  -->  1111010
Y  -->  1111000
B  -->  111001

Binary coding tree:
---b
----o
----8
----6
-----&
-----7
----i
-----N
-----u
-----t
-----*
-----(
-----p
-----n
-----y
-----U
-----^
-----\n
-----9
-----m
------g
------P
------1
------B
------space
-------f
-------tabulation
-------Y
-------!
-------I
--------j
--------G
-------T
-------w
-------e
--------r
--------d

Read text from file ./Tests/test4.txt
Input text:

Encoded text:

Character codes:
```

Binary coding tree:

Приложение В. Исходный код программы

main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "fanoshannontree.h"


using std::cout;
using std::cin;
using std::cerr;


std::string readText(std::istream& in);

std::string readText(std::ifstream& in);


int main(int argc, char *argv[]) {
    setlocale(LC_ALL, "");

    if (argc < 3) {
        std::string text;
        std::string result;

        if (argc == 2) {
            std::string fileName = argv[1];
            result += "Read text from file " + fileName + "\n";

            std::ifstream inputFile(fileName, std::ios::in);
            text = readText(inputFile);
            inputFile.close();
        }
        else {
            cout << "Enter text: ";
            text = readText(cin);
        }

        std::ofstream resultFile("result.txt", std::ios::app);

        FanoShannonTree* tree = new FanoShannonTree();
        result += "Input text:\n" + text + "\n\n";
        result += tree->encode(text) + "\n\n\n";
        delete tree;

        cout << result;
        resultFile << result;
```

```
        resultFile.close();
    }
    else {
        cerr << "Error: incorect console's arguments\n";
    }

    cout << "End programm\n";
    return 0;
}


std::string readText(std::istream& in) {
    std::string result;
    std::string oneLineStr;
    while (getline(in, oneLineStr)) {
        result += oneLineStr;
        result += "\n";
    }
    result = result.substr(0, result.size() - 1);

    return result;
}


std::string readText(std::ifstream& in) {
    if (!in.is_open()) {
        cerr << "Error: incorrect file name!\n";
        return "";
    }

    std::string result;
    std::string oneLineStr;
    while (getline(in, oneLineStr)) {
        result += oneLineStr;
        result += "\n";
    }
    result = result.substr(0, result.size() - 1);

    return result;
}
```

encodetree.h

```
#ifndef ENCODETREE_H
#define ENCODETREE_H

#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <algorithm>
```

```
struct Symbol
{
    /*
     * Структура для хранения символа и его 'веса'
    */

    char data_;
    size_t weight_;

    Symbol(char data = 0, size_t weight = 0) :
        data_(data), weight_(weight) { }
};


class EncodeTree
{
    /*
     * Базовый абстрактный класс дерева, реализующий методы для работы с
     * бинарным деревом кодирования. Экземпляр класса является узлом дерева.
    */

public:
    EncodeTree();
    EncodeTree(const Symbol& data);
    virtual ~EncodeTree();

    //Getter's - методы
    const EncodeTree* getLeft() const;
    const EncodeTree* getRight() const;
    const EncodeTree* getParent() const;
    const Symbol& getData() const;

    //Setter's - методы
    void setLeft(EncodeTree* left);
    void setRight(EncodeTree* right);
    void setParent(EncodeTree* parent);
    void setData(Symbol data);

    //Метод объединения левого и правого поддеревьев в узел
    void concat(size_t weight, EncodeTree* left, EncodeTree* right);

    //Метод для кодирования строки с помощью бинарного дерева кодирования
    std::string encode(const std::string& message);

protected:
    //Два виртуальных метода для создания дерева. В зависимости от алгоритма,
    //класс-наследник реализует эти методы

    virtual void createEncodeTree(const std::string& message) = 0;

    virtual void createEncodeTree(std::map<char, size_t>& symbolMap,
                                  std::vector<char>& symbolVector,
                                  size_t left, size_t right, size_t sum) = 0;
```

```
    //Метод поиска листа дерева с заданными данными
    const EncodeTree* findSymbol(char data) const;

    //Метод записи дерева в строку в виде уступчатого списка
    void printTree(std::string& result, size_t level = 0) const;

    //Метод создания строки кодов символов
    std::string createCodeSymbols(const std::string& message) const;

private:
    Symbol data_;
    EncodeTree* parent_;
    EncodeTree* left_;
    EncodeTree* right_;
};


#endif // ENCODETREE_H
```

encodetree.cpp

```
#include "encodetree.h"


EncodeTree::EncodeTree()
{
    parent_ = nullptr;
    left_ = nullptr;
    right_ = nullptr;
}


EncodeTree::EncodeTree(const Symbol& data)
{
    /*
     * Конструктор создания листа дерева, принимает на вход символ
     * с его "весом".
    */

    this->data_ = data;
}


EncodeTree::~EncodeTree()
{
    if (left_ != nullptr)
        delete left_;

    if (right_ != nullptr)
        delete right_;
}
```

```
const EncodeTree *EncodeTree::getLeft() const
{
    return left_;
}


const EncodeTree *EncodeTree::getRight() const
{
    return right_;
}


const EncodeTree *EncodeTree::getParent() const
{
    return parent_;
}


const Symbol& EncodeTree::getData() const
{
    return data_;
}


void EncodeTree::setLeft(EncodeTree *left)
{
    left_ = left;
}


void EncodeTree::setRight(EncodeTree *right)
{
    right_ = right;
}


void EncodeTree::setParent(EncodeTree *parent)
{
    parent_ = parent;
}


void EncodeTree::setData(Symbol data)
{
    data_ = data;
}


void EncodeTree::concat(size_t weight, EncodeTree* left, EncodeTree* right)
{
    /*
     * Метод создания узла дерева, принимает на вход "вес"
```

```
     * поддеревьев, левое и правое поддеревья.
    */

    this->data_.weight_ = weight;
    this->left_ = left;
    this->right_ = right;
}


std::string EncodeTree::encode(const std::string &message)
{
    /*
     * Функция кодирования текста.
     * Принимает на вход текст, возвращает строку-результат,
     * содержащую закодированный текст, коды символов, представление дерева.
    */

    this->createEncodeTree(message);

    std::string code;
    std::string path;

    for (auto elem : message) {
        const EncodeTree* node = this->findSymbol(elem);

        while (node->parent_ != nullptr) {
            if (node->parent_->left_ == node) {
                path = "0" + path;
            }
            else {
                path = "1" + path;
            }
            node = node->parent_;
        }
        code += path;
        path.clear();
    }

    std::string res = "Encoded text:\n";
    res += code;

    res += "\nCharacter codes:\n";
    res += this->createCodeSymbols(message);

    std::string resTree = "\nBinary coding tree: \n";
    this->printTree(resTree);
    res += resTree;
    return res;
}


const EncodeTree *EncodeTree::findSymbol(char data) const
{
```

```
    /*
     * Функция поиска символа в дереве, принимает указатель на корень и символ,
     * возвращает указатель на лист, который содержит этот символ, либо если
     * символ отсутствует в дереве - nullptr.
    */

    if (this->left_ == nullptr && this->right_ == nullptr) {
        if (this->data_.data_ == data) {
            return this;
        }
    }
    if (this->left_ != nullptr) {
        const EncodeTree* buffer = this->left_->findSymbol(data);
        if (buffer != nullptr) {
            return buffer;
        }
    }
    if (this->right_ != nullptr) {
        const EncodeTree* buffer = this->right_->findSymbol(data);
        if (buffer != nullptr) {
            return buffer;
        }
    }
    return nullptr;
}


void EncodeTree::printTree(std::string &result, size_t level) const
{
    /*
     * Функция записи дерева в строку. Принимает на вход ссылку на
     * строку-результат и уровень в дереве.
     *
     * Записывает дерево в строку ввиде уступчатого списка.
    */

    if (this->right_ == nullptr && this->left_ == nullptr){
        for (size_t i = 0; i < level; ++i){
            result += "-";
        }

        if (this->data_.data_ == ' ') {
            result += "space";
        }
        else if (this->data_.data_ == '\n') {
            result += "\\n";
        }
        else if (this->data_.data_ == '\t') {
            result += "tabulation";
        }
        else {
            result.push_back(this->data_.data_);
        }
```

```
        result.push_back('\n');
    }
    if (this->left_){
        this->left_->printTree(result, level + 1);
    }
    if (this->right_){
        this->right_->printTree(result, level + 1);
    }
}


std::string EncodeTree::createCodeSymbols(const std::string &message) const
{
    /*
     * Функция определения кодов символов по дереву кодирования, текст и
     * возвращает строку-результат.
     *
     * 1) Создается массив уникальных символов. Алгоритм работает за O(n*m), где n -
     * текста, m - количество уникальных символов. т.к char может содержать не более
2^8
     * символов, можно считать, что алгоритм работает за O(n)
     *
     * 2) В строку результат записывается код символа, который сохраняется в
     * переменной path при проходе по дереву
    */

    std::vector<char> symbolVector;

    std::string path;
    std::string res;

    for (auto elem : message){
        if (find(symbolVector.begin(), symbolVector.end(), elem) == symbolVector.end()
)
            symbolVector.push_back(elem);
    }

    for (size_t i = 0; i < symbolVector.size(); ++i){
        // Переходим в ту часть дерева, где находится нужный символ
        const EncodeTree* node = this->findSymbol(symbolVector[i]);
        // При подъеме от листа к корню если переходим влево - то
        // добавляем в массив 0, если вправо - 1
        while (node->parent_ != nullptr) {
            if (node->parent_->left_ == node) {
                path = "0" + path;
            }
            else {
                path = "1" + path;
            }

            node = node->parent_;
        }
```

```
        if (symbolVector[i] == '\n') {
            res += "\\n";
        }
        else if (symbolVector[i] == ' ') {
            res += "space";
        }
        else if (symbolVector[i] == '\t') {
            res += "tabulation";
        }
        else {
            res.push_back(symbolVector[i]);
        }

        res += "  -->  " + path + "\n";
        path.clear();
    }

    return res;
}
```

fanoshannontree.h

```
#ifndef FANOSHANNONTREE_H
#define FANOSHANNONTREE_H

#include "encodetree.h"


class FanoShannonTree : public EncodeTree
{
    /*
     * Класс, реализующий алгоритм кодирования Фано-
     * Шеннона с помощью бинарного дерева кодирования.
    */

public:
    FanoShannonTree() = default;
    FanoShannonTree(const Symbol& data);
    ~FanoShannonTree() = default;

private:
    //Два метода создания дерева кодирования с помощью алгоритма
    //Фано-Шеннона
    void createEncodeTree(const std::string& message);

    void createEncodeTree(std::map<char, size_t>& symbolMap,
                          std::vector<char>& symbolVector,
                          size_t left, size_t right, size_t sum);

    //Статический метод для получения середины массива символов,
    //отсортированных по невозрастанию, по их 'весу'
    static size_t getMiddle(std::map<char, size_t>& symbolMap,
```

```
                            std::vector<char>& symbolVector,
                            size_t left, size_t sum, size_t& leftSum,
                            size_t& rightSum);
};

#endif // FANOSHANNONTREE_H
```

fanoshannontree.cpp

```
#include "fanoshannontree.h"


FanoShannonTree::FanoShannonTree(const Symbol &data) : EncodeTree(data) { }


void FanoShannonTree::createEncodeTree(const std::string &message)
{
    /*
     * Функция, для создания бинарного дерева кодирования, принимает на вход
     * текст.
     *
     * 1) Создаются словарь и массив символов. В словарь заносится частота
     * встречаемости каждого символа в тексте.
     * В массив заносятся уникальные символы.
     *
     * 2) Массив сортируется по убыванию "веса" символов.
     *
     * 3) Случай, когда текст содержит один символ обрабатывается отдельно.
     *
     * 4) Если в тексте больше 1 символа, вызывается фунция создания дерева,
     * которая создает поддеревья из подмассивов.
    */

    std::map<char, size_t> symbolMap;
    std::vector<char> symbolVector;
    for (auto elem : message) {
        if (symbolMap.count(elem) == 0) {
            symbolMap.insert(std::make_pair(elem, 1));
            symbolVector.push_back(elem);
        }
        else {
            symbolMap[elem] += 1;
        }
    }

    std::sort(symbolVector.begin(), symbolVector.end(),
              [&symbolMap] (char first, char second) {
        return symbolMap[first] > symbolMap[second]; });

    if (symbolVector.size() == 1) {
        char data = symbolVector[0];
        Symbol symbol(data, symbolMap[data]);
        FanoShannonTree* leftTree = new FanoShannonTree(symbol);
```

```
        this->setLeft(leftTree);
        this->setData(Symbol(0, symbolMap[data]));
        leftTree->setParent(this);
    }

    return createEncodeTree(symbolMap, symbolVector, 0,
                            symbolVector.size(), message.length());
}


void FanoShannonTree::createEncodeTree(std::map<char, size_t> &symbolMap,
                                       std::vector<char> &symbolVector,
                                       size_t left, size_t right, size_t sum)
{
    /*
     * Функция, для создания бинарного дерева кодирования, принимает на вход
     * словарь символов, отсортированный по убыванию массив, левый и правый индекс в
массиве,
     * сумму "весов" символов.
     *
     * 1) Если левый индекс больше либо равен правому - возвращает nullptr.
     *
     * 2) Если между левым и правым индексами один элемент - создаем лист.
     *
     * 3) В ином случае разделяем массив на деревья, примерно равные по весу,
     * рекурсивно создаем левое и правое поддеревья.
    */

    if (left >= right) {
        return;
    }
    if (right == left + 1) {
        char data = symbolVector[left];
        Symbol symbol(data, symbolMap[data]);
        this->setData(symbol);
        return;
    }

    size_t leftSum = 0;
    size_t rightSum = 0;
    size_t middle = getMiddle(symbolMap, symbolVector, left, sum, leftSum, rightSum);

    FanoShannonTree* leftTree = new FanoShannonTree();
    leftTree->createEncodeTree(symbolMap, symbolVector, left,
                               middle + 1, leftSum);

    FanoShannonTree* rightTree = new FanoShannonTree();
    rightTree->createEncodeTree(symbolMap, symbolVector, middle + 1,
                                right, rightSum);

    this->concat(sum, leftTree, rightTree);
    leftTree->setParent(this);
    rightTree->setParent(this);
```

```
}


size_t FanoShannonTree::getMiddle(std::map<char, size_t> &symbolMap,
                                  std::vector<char> &symbolVector,
                                  size_t left, size_t sum, size_t &leftSum,
                                  size_t &rightSum)
{
    /*
     * Функция получения середины массива символов, принимает на вход
     * словарь символов, отсортированный по убыванию массив, левый индекс в массиве,
     * ссылки на сумму весов слева и справа от середины, возвращает индекс середины.
    */

    size_t middle = left;

    leftSum = symbolMap[symbolVector[middle]];
    rightSum = sum - leftSum;

    long delta = static_cast<long>(leftSum) - static_cast<long>(rightSum);

    while (delta + static_cast<long>(symbolMap[symbolVector[middle+1]]) < 0) {
        ++middle;
        char data = symbolVector[middle];
        leftSum += symbolMap[data];
        rightSum -= symbolMap[data];
        delta = static_cast<long>(leftSum) - static_cast<long>(rightSum);
    }
    return middle;
}
```